# Wandering in the Labyrinth of Thinking

**– a minimalist cognitive architecture combining reinforcement learning and deep learning**

甄景贤 (King-Yin Yan), Ben Goertzel, and Juan Carlos Kuri Pinto

General.Intelligence@Gmail.com

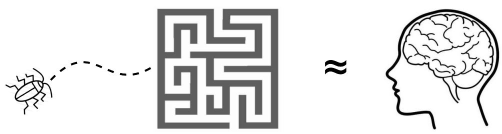
**Abstract.** This is the first of a series of papers, introducing a minimalist cognitive architecture based on reinforcement learning and deep learning. The system consists of an iterative function whose role is analogous to the consequence operator ($\vdash$) in logic. Mathematically it is a Hamiltonian system, whose Lagrangian corresponds to the value of "desires" or "rewards" for the intelligent agent. Techniques of classical logic-based AI can be transferred to the neural setting, the topic of our 2nd paper.

In order to understand our main theory [6], most of the present paper can be ignored, except for §0 and a basic understanding of reinforcement learning (eg. my tutorial [7]).

Most of this paper is background information, eg. the "Hamiltonian theory" is already well-known and is not our original contribution.

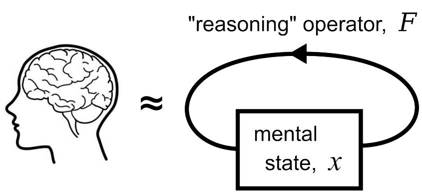
# 0 Main idea

The **metaphor** here is that of reinforcement learning controlling an autonomous agent to navigate the maze of "thoughts space", seeking the optimal path:



(1)

The main idea is to regard "thinking" as a **dynamical system** operating on **mental states**:




(2)

For example, a mental state could be the following set of propositions:

- I am in my room, writing a paper for AGI-17.
- I am in the midst of writing the sentence, "I am in my room, ..."

- I am about to write a gerund phrase "writing a paper..."

Thinking is the process of **transitioning** from one mental state to another. Even as I am speaking now, I use my mental state to keep track of where I am at within the sentence's syntax, so that I can structure sentences grammatically.

The following 3 theories are actually synonymous:

- in artificial intelligence, **reinforcement learning (RL)**
- in operations research, **dynamic programming**
- in modern control theory, the **state space** description

# 1 Control theory / dynamical systems theory

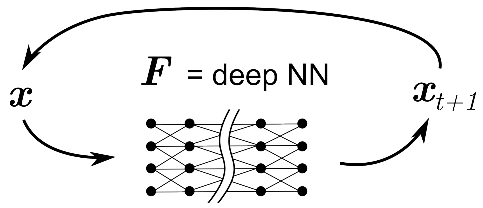
The cognitive state is a vector $\boldsymbol{x} \in \mathbb{X}$ where $\mathbb{X}$ is the space of all possible cognitive states, the reasoning operator $\boldsymbol{F}$ is an **endomorphism** (an **iterative map**) $\mathbb{X} \to \mathbb{X}$.

Mathematically this is a **dynamical system** that can be defined by:

$$\boxed{\text{discrete time}} \qquad \boldsymbol{x}_{t+1} = \boldsymbol{F}(\boldsymbol{x}_t) \tag{3}$$

$$\boxed{\text{continuous time}} \qquad \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) \tag{4}$$

In our cognitive architecture design, $\boldsymbol{F}$ is implemented as a deep neural network (the word "deep" simply means "many layers"):




(5)

A **neural network** is a non-linear operator with many parameters (called "weights"):

each layer's **weight** matrix      total # of layers

$$F(\boldsymbol{x}) = \sigma(W_1 \sigma(W_2 ... \sigma(W_L\ \boldsymbol{x}))) \tag{6}$$

$\sigma$ is a sigmoid-shaped non-linear function, applied component-wise to the vectors.

If continuous-time, $\boldsymbol{f}$ can also be implemented as neural network, but $\boldsymbol{f}$ and $\boldsymbol{F}$ are different in nature, they are related by: $\boldsymbol{x}(t+1) = \boldsymbol{F}(\boldsymbol{x}(t))$. For ease of discussion, sometimes I mix discrete-time and continuous-time notations.

A **control system** is a dynamical system added with the control vector $\boldsymbol{u}(t)$:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \tag{7}$$

The goal of control theory is to find the optimal $\boldsymbol{u}^*(t)$ function, such that the system moves from the initial state $\boldsymbol{x}_0$ to the terminal state $\boldsymbol{x}_\perp$.

A typical control-theory problem is described by:

$$\boxed{\text{state equation}} \qquad \dot{\boldsymbol{x}}(t) = \boldsymbol{f}[\boldsymbol{x}(t), \boldsymbol{u}(t), t] \tag{8}$$

$$\boxed{\text{boundary condition}} \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \, , \; \boldsymbol{x}(t_\perp) = \boldsymbol{x}_\perp \tag{9}$$

$$\boxed{\text{objective function}} \qquad J = \int_{t_0}^{t_\perp} L[\boldsymbol{x}(t), \boldsymbol{u}(t), t] dt \tag{10}$$

and we seek the optimal control $\boldsymbol{u}^*(t)$.

According to control theory, the condition for **optimal path** is given by the Hamilton-Jacobi equation:

$$\boxed{\text{Hamilton-Jacobi equation}} \qquad 0 = \frac{\partial J^*}{\partial t} + \min_u H \tag{11}$$

After the next section I will explain the meaning of $J$, $L$ and $H$.

# 2 Reinforcement learning / dynamic programming

**Reinforcement learning** is a branch of machine learning that is particularly suitable for controlling an **autonomous agent** who interacts with an **environment**. It uses **sensory perception** and **rewards** to continually modify its **behavior**. The exemplary image you should invoke in mind is that of a small insect that navigates a maze looking for food and avoiding predators: 

A reinforcement learning system consists of a 4-tuple:

$$\boxed{\text{reinforcement learning system}} = (\boldsymbol{x} \in \text{States}, \boldsymbol{u} \in \text{Actions}, R = \text{Rewards}, \pi = \text{Policy}) \tag{12}$$

For details readers may see my *Reinforcement learning tutorial* [7].

$U$ is the total rewards of a sequence of actions:

$$\overset{\text{total value of state 0}}{U(\boldsymbol{x}_0)} = \sum_t \overset{\text{reward at time } t}{R(\boldsymbol{x}_t, \boldsymbol{u}_t)} \tag{13}$$

For example, the value of playing a chess move is not just the immediate reward of that move, but includes the consequences of playing that move (eg, greedily taking a pawn now may lead to checkmate 10 moves later). Or, faced with delicious food, some people may choose not to eat, for fear of getting fat.

The central idea of **Dynamic programming** is the **Bellman optimality condition**. Richard Bellman in 1953 proposed this formula, while he was working at RAND corporation, dealing with operations research problems.

The **Bellman condition** says: "if we cut off a tiny bit from the endpoint of the optimal path, the remaining path is still an optimal path between the new endpoints."

$$\boxed{\text{Bellman equation}} \quad \overbrace{U^*(\boldsymbol{x})}^{\text{value of entire path}} = \max_{\boldsymbol{u}} \{ \overbrace{R(\boldsymbol{u})}^{\text{reward of choosing } \boldsymbol{u} \text{ at current state}} + \overbrace{U^*(\boldsymbol{x}_{t+1})}^{\text{value of rest of path}} \} \tag{14}$$

This seemingly simple formula is the entire content of dynamic programming; What it means is that: When seeking the path with the best value, we cut off a bit from the path, thus reducing the problem to a smaller problem; In other words, it is a **recursive relation** over time.

In AI reinforcement learning there is an oft-employed trick known as $Q$-learning. $Q$ value is just a variation of $U$ value; there is a $U$ value for each state, and $Q$ is the **decomposition** of $U$ by all the actions available to that state. In other words, $Q$ is the utility of doing action $\boldsymbol{u}$ in state $\boldsymbol{x}$. The relation between $Q$ and $U$ is:

$$U(\boldsymbol{x}) = \max_{\boldsymbol{u}} Q(\boldsymbol{x}, \boldsymbol{u}) \tag{15}$$

The advantage of $Q$ is the ease of learning. We just need to learn the value of actions under each state. This is so-called "**model free learning**".

Under the reinforcement learning framework, intelligence is decomposed into **thinking** and **learning**:

- **Thinking** means finding the optimal trajectory of $\boldsymbol{x}$ according to the **knowledge** stored in the deep NN. This is achieved using the Bellman equation to calculate $\boldsymbol{u}^*$. The trajectory of $\boldsymbol{x}$ is constrained by the deep NN (in other words, the system must think in accordance with **correct knowledge**). While thinking, the deep NN stays **constant**.
- **Learning** means learning the weights in the deep NN. Changing $W$ changes $\boldsymbol{F}$, which determines the state equation (3), so the entire system becomes a new one. In other words, deep NN learning is a kind of **second-order learning**: Consider 2 systems $\boldsymbol{F}$ and $\boldsymbol{F} + \epsilon\hat{\boldsymbol{F}}$, after many trials of thinking with different premises, if the average reward is higher in the altered system, $\boldsymbol{F}$ will learn towards the $\hat{\boldsymbol{F}}$ direction.

### Prior art

The minimalist architecture based on reinforcement learning has been proposed by Itimar Ariel from Israel, in 2012 [2], and I also independently proposed in 2016 (precursor of this paper). The prestigious researcher of signal processing, Simon Haykin, recently also used the "RL + memory" design, cf. his 2012 book *Cognitive dynamic systems* [3]. Vladimir Anashin in the 1990's also proposed this kind of cognitive architecture [1]. Perhaps there exist other precedents, eg: [4].

# 3 Connections with the Hamiltonian

In **reinforcement learning**, we are concerned with two quantities:

- $R(\boldsymbol{x}, \boldsymbol{u})$ = **reward** of doing action $\boldsymbol{u}$ in state $\boldsymbol{x}$
- $U(\boldsymbol{x})$ = **utility** or **value** of state $\boldsymbol{x}$

Simply put, **utility** is the integral of instantaneous **rewards** over time:

$$\boxed{\text{utility } U} = \int \boxed{\text{reward } R} \; dt \tag{16}$$

In **control-theoretic** parlance, it is usually defined the **cost functional**:

$$\boxed{\text{cost} \quad J} = \int L dt + \Phi(\boldsymbol{x}_\perp) \tag{17}$$

where $L$ is the **running cost**, ie, the cost of making each step; $\Phi$ is the **terminal cost**, ie, the value when the terminal state $\boldsymbol{x}_\perp$ is reached.

In **analytical mechanics** $L$ is known as the **Lagrangian**, and the time-integral of $L$ is called the **action**:

$$\boxed{\text{action} \quad S} = \int L dt \tag{18}$$

Hamilton's **principle of least action** says that $S$ always takes the **stationary value**, ie, the $S$ value is extremal compared with neighboring trajectories.

The **Hamiltonian** is defined as $H = L + \dfrac{\partial J^*}{\partial \boldsymbol{x}} \boldsymbol{f}$, which came from the method of **Lagrange multipliers**. For details please refer to my *Control theory tutorial*[**?**].

All these are the same thing, so there is this correspondence:

| **Reinforcement learning** | **Control theory** | **Analytical mechanics** |
|---|---|---|
| utility or value $U$ | cost $J$ | action $S$ |
| instantaneous reward $R$ | running cost | Lagrangian $L$ |
| action $a$ | control $u$ | (external force?) |

(19)

Interestingly, the reward $R$ corresponds to the **Lagrangian** in physics, whose unit is "energy"; In other words, "desires" or "happiness" appear to be measured by units of "energy", this coincides with the idea of "positive energy" in pop psychology. Whereas, long-term value is measured in units of [energy × time].

This correspondence of these 3 theories is explained in detail in Daniel Liberzon's book [5]. While this correspondence has very interesting philosophical implications, it may not be very useful in practice: the traditional AI system is discrete-time; converting it to continuous-time seems to increase the computational burden, while it is unclear what advantages this might bring....

The equation of motion for the continuous-time case is the famed **Hamilton-Jacobi-Bellman equation**:

$$\boxed{\text{Hamilton-Jacobi-Bellman}} \qquad 0 = \frac{\partial U^*}{\partial t} + \min_u H \tag{20}$$

of which the **Schrödinger equation** is a special case.

# 4 Future directions

- **Relation to logic-based AI:** In the system's state equation (3), $\boldsymbol{F}$ is free to change ($\boldsymbol{F}$ represents learned knowledge). In other words, the entire system almost has no structure. Searching for a candidate $\boldsymbol{F}$ in the infinite-dimensional function space is impractical, so we need to introduce the structure of logic-based AI into this system, such that the search space for $\boldsymbol{F}$ is reduced. In machine learning, this is known as **inductive bias**, the *sine qua non* of speeding up learning. This will be addressed in our 2nd paper *A bridge between logic and neural* [6].
- **Memory:** In this minimal architecture there is no episodic memory, this will be dealt with by our 3rd paper, *The structure of memory* [8].

# References


1. Vladimir Anashin and Andrei Khrennikov. *Applied algebraic dynamics*. de Gruyter, 2009.
2. Itamar Arel. *Deep reinforcement learning as Foundations for Artificial Intelligence*, chapter 6, pages 89–102. Atlantis Press, 2012.
3. Simon Haykin. *Cognitive dynamic systems*. Cambridge Univ Press, 2012.
4. Vladimir Ivancevic and Tijana Ivancevic. *Geometrical dynamics of complex systems: a unified modeling approach to physics, control, biomechanics, neurodynamics and psycho-socio-economical dynamics*. Springer, 2006.
5. Daniel Liberzon. *Calculus of variations and optimal control theory: a concise introduction*. Princeton Univ Press, 2012.
6. King Yin Yan. A bridge between logic and neural. (to be submitted AGI-2017).
7. King Yin Yan. Reinforcement learning tutorial `https://drive.google.com/file/d/0Bx3_S9SExak-X29uazI1YnhoNFU/view`.
8. King Yin Yan. The structure of memory. (to be submitted AGI-2017).